

After watching this lecture you will know how FSMs can be implemented in VHDL. We will further present best-practices regarding the coding style and discuss some related pitfalls.

HWMoD
WS25

FSM Imple-
mentation
Implementation
Stops
Example

Hardware Modeling [VU] (191.011) – WS25 – Finite-State Machine Implementation (in VHDL)

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

As already mentioned in the FSM modelling lecture, FSM models adhering to our modelling approach can be implemented in VHDL by performing a sequence of simple steps. This conversion from an FSM model to VHDL code works for all FSMs you will encounter in this and future courses. It is one of the key take-ways from this course! Hence, make sure that you can perform such conversions yourself.

Implementing FSMs in VHDL

HWMoD
WS25

FSM Imple-
mentation
Implementation
Steps
Example

- Conversion of FSM model to VHDL code follows simple steps

The first step in the conversion process is to create an enumeration type containing all the abstract FSM state names used in the graph model.

Implementing FSMs in VHDL

HWMoD
WS25

FSM Imple-
mentation
Implementation
Steps
Example

- Conversion of FSM model to VHDL code follows simple steps
 - 1 Create an enumeration type for the FSM state

To represent the state register we need another custom type in the form of the state register record type. This record consists of all the state variables used in the model. These can be found easily, by gathering the left-hand sides of all state update assignments. Furthermore, the state register also contains an element that identifies the current abstract FSM state of the graph model. For this purpose we use the enumeration type declared in step one.

Implementing FSMs in VHDL

HWMoD
WS25

FSM Imple-
mentation
Implementation
Steps
Example

- Conversion of FSM model to VHDL code follows simple steps
 - 1 Create an enumeration type for the FSM state
 - 2 Create a record type for the state register

Next, we need to declare exactly two signals of the state register record type. One represents the current value of the state register, usually named s . Note that this is exactly why we prefixed the current state register elements with $s.$ and the next state register elements with $s'.$ in the modelling lecture. This notation ensures, that accesses to the state register in state updates and output actions are already syntactically valid accesses to elements of a VHDL record type. This allows us to later directly use the expressions of the graph model in our code. The other signal represents the output of the next-state logic which holds the next value of the state register. Therefore, we refer to it as s_next .

Implementing FSMs in VHDL

HWMod
WS25

FSM Imple-
mentation
Implementation
Steps
Example

- Conversion of FSM model to VHDL code follows simple steps
 - 1 Create an enumeration type for the FSM state
 - 2 Create a record type for the state register
 - 3 Create signals for the current (s) and next (s_next) register value

In the fourth step the state register is implemented using a synchronous process as introduced in the lecture about sequential circuit elements. As an example will demonstrate shortly, this process is very simple and virtually always the same for all FSMs.

Implementing FSMs in VHDL

HWMoD
WS25

FSM Imple-
mentation
Implementation
Steps
Example

- Conversion of FSM model to VHDL code follows simple steps
 - 1 Create an enumeration type for the FSM state
 - 2 Create a record type for the state register
 - 3 Create signals for the current (s) and next (s_next) register value
 - 4 Implement the state register in a sync. process

Finally, in the fifth step, the next-state and output logic are implemented using one or two combinational processes. While these processes contain the most code, as they implement the actual functionality of the FSM, their basic structure is the same for all FSM implementations, making their implementation quite straight forward. If you use our modelling approach, you can write down these processes in an almost mechanical manner.

Implementing FSMs in VHDL

HWMod
WS25

FSM Imple-
mentation
Implementation
Steps
Example

- Conversion of FSM model to VHDL code follows simple steps
 - 1 Create an enumeration type for the FSM state
 - 2 Create a record type for the state register
 - 3 Create signals for the current (s) and next (s_next) register value
 - 4 Implement the state register in a sync. process
 - 5 Implement the next-state and output logic in comb. process(es)

Depending on whether you use one or two processes for the next-state and output logic we refer to an implementation as using the 2, respectively 3, process method. However, since the next-state and output functions usually share a significant amount of logic, the two-process method is typically preferable as it is more readable and reduces redundant code. It is therefore also easier to maintain, modify and extend. We therefore strongly recommend you to exclusively use the 2-process method.

Implementing FSMs in VHDL

HWMoD
WS25

FSM Imple-
mentation
Implementation
Steps
Example

- Conversion of FSM model to VHDL code follows simple steps
 - 1 Create an enumeration type for the FSM state
 - 2 Create a record type for the state register
 - 3 Create signals for the current (s) and next (s_next) register value
 - 4 Implement the state register in a sync. process
 - 5 Implement the next-state and output logic in comb. process(es)
- 2- vs. 3-process method
 - Overlap of next-state and output logic ⇒ 2-process usually more readable
 - Recommendation: Use the 2-process method

Finite-State Machine Implementation (in VHDL)

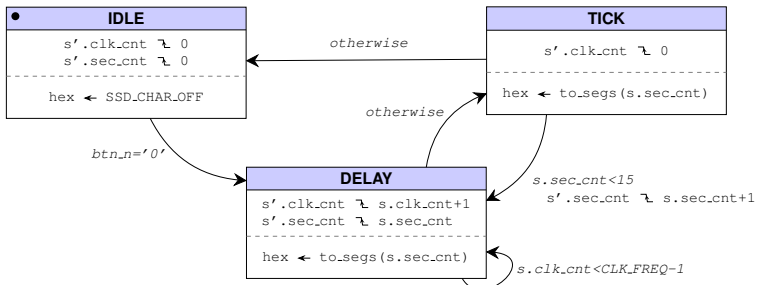
Example

Example: FSM Types



Let us now demonstrate these FSM implementation steps at the hand of the `advanced_timer` example for which we already created a model. As a reminder, this model is shown on the slide.

Example: FSM Types



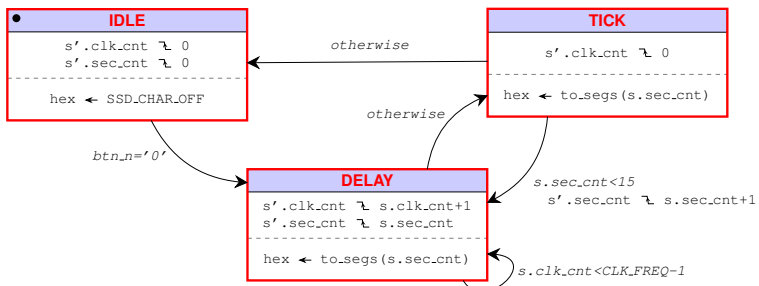


The first step is to declare an enumeration type for the abstract states of the FSM. As already mentioned, the values of this type are the names of the FSM's abstract states. In this case, the FSM has three states named IDLE, DELAY and TICK.

Example: FSM Types

1 Create an enumeration type for the FSM state

```
20 type fsm_state_t is (IDLE, DELAY, TICK);
```



Finite-State Machine Implementation (in VHDL)

Example

Example: FSM Types

2 Create a record type for the state register

```
21 type fsm_state_t is (IDLE, DELAY, TICK);
22 type state_reg_t is record
23     state : fsm_state_t;
```

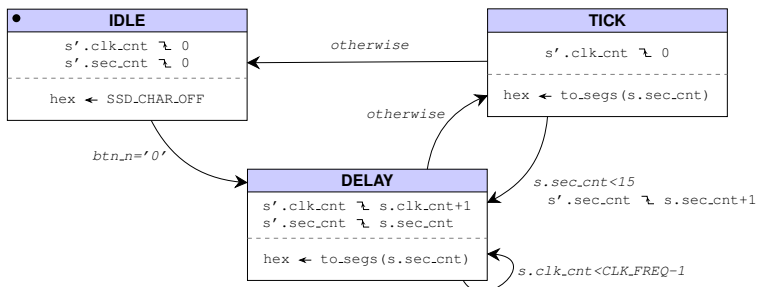


Next, we declare the record type for the state register. As mentioned, this type shall always contain an element for the FSM's abstract state. Naturally, as shown on the slide, this element is of the previously declared enumeration type. Now we have to determine which other elements the state register must contain. We do that by gathering the target elements of all state updates, regardless of whether they can be found in a state node or alongside a transition edge.

Example: FSM Types

2 Create a record type for the state register

```
20 type fsm_state_t is (IDLE, DELAY, TICK);
21 type state_reg_t is record
22     state : fsm_state_t;
```



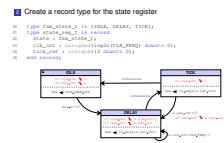
HWMMod
WS25

FSM Implementation
Implementation Steps
Example
FSM Types
State Register
Comb. Logic

Finite-State Machine Implementation (in VHDL)

Example

Example: FSM Types

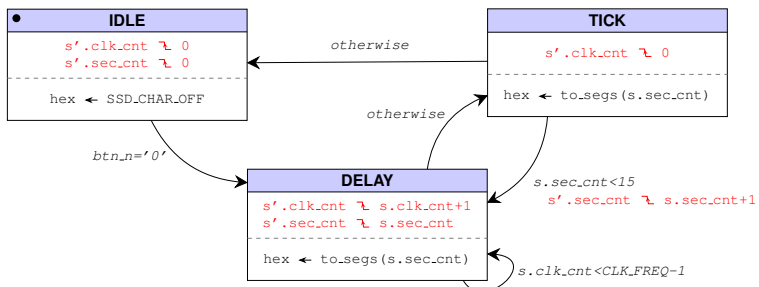


On the slide we colored the left-hand sides of all state updates contained in our model red.

Example: FSM Types

2 Create a record type for the state register

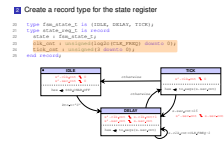
```
20 type fsm_state_t is (IDLE, DELAY, TICK);
21 type state_reg_t is record
22     state : fsm_state_t;
23     clk_cnt : unsigned(log2c(CLK_FREQ) downto 0);
24     tick_cnt : unsigned(3 downto 0);
25 end record;
```



Finite-State Machine Implementation (in VHDL)

Example

Example: FSM Types

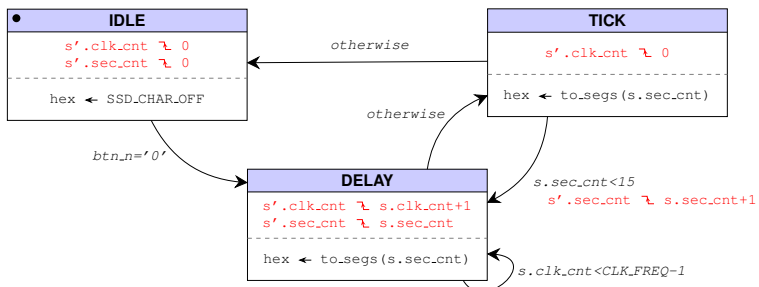


We can clearly observe that the state register requires two counters. One of them must support counting up to the clock frequency, while the other one must be able to store values up to fifteen for counting the passed seconds. The data type information of state register elements is either taken from the behavior description of the respective FSM, or from the model, by looking at the state updates and conditions involving a specific element.

Example: FSM Types

2 Create a record type for the state register

```
20 type fsm_state_t is (IDLE, DELAY, TICK);
21 type state_reg_t is record
22     state : fsm_state_t;
23     clk_cnt : unsigned(log2c(CLK_FREQ) downto 0);
24     tick_cnt : unsigned(3 downto 0);
25 end record;
```



In the third step we have to declare the signals for the state register and the output of the next-state logic. While we already mentioned that these are really the only two signals you need to declare when implementing an FSM, we want to illustrate why this is the case.

Example: State Register

3 Create signals for the current (s) and next (s_next) register value

```
18 architecture arch of advanced_timer is
19 -- enum and record declarations
```

Finite-State Machine Implementation (in VHDL)

Example

Example: State Register

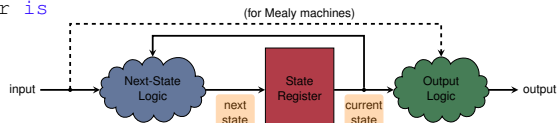


For that, recall the basic structure of a finite state machine as shown on the slide. If we think about this structure as being a VHDL entity the `input` and `output` are clearly ports of it. Therefore, since the combinational functions and the register can be implemented in a single process each, there are only two internal interfaces for which we require dedicated signals. One is at the output of the state register, which provides the current value stored in the register to the next-state and output logic, while the other one is at the state register's input and provides its next value. Both of these interfaces are highlighted on the slide.

Example: State Register

3 Create signals for the current (s) and next (s_nxt) register value

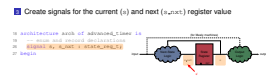
```
18 architecture arch of advanced_timer is  
19 -- enum and record declarations
```



Finite-State Machine Implementation (in VHDL)

Example

Example: State Register

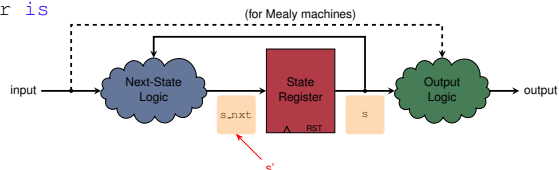


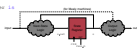
By adding these two signal declarations the declarative part of our FSM implementation's architecture is complete and we can proceed with the next step. Note that the signal `s_nxt` is somewhat related to what we referred to as `s'` so far, as it is the value the state registers will have after the next rising clock edge.

Example: State Register

3 Create signals for the current (s) and next (s_nxt) register value

```
18 architecture arch of advanced_timer is
19   -- enum and record declarations
26   signal s, s_nxt : state_reg_t;
27 begin
```





In the fourth step, we implement the state register. For that we assume that our FSM is clocked by a signal called `clk` and that it features an asynchronous, active-low, reset.

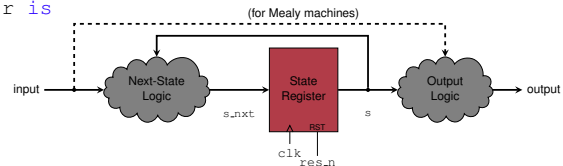
Example: State Register

4 Implement the state register in a sync. process

```

18 architecture arch of advanced_timer is
19   -- enum and record declarations
20   signal s, s_nxt : state_reg_t;
21 begin

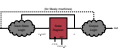
```



```

18 architecture arch of advanced_timer is
19   -- enum and record declarations
20   signal s, s_nxt : state_reg_t;
21 begin
22   state_reg : process(clk, res_n)
23   begin
24     if res_n = '0' then
25       s <= (state => IDLE, others => (others => '0'));
26     elsif rising_edge(clk) then
27       s <= s_nxt;
28     end if;
29   end process;
30 end arch;

```



Since we all watched the video about describing sequential logic in VHDL we of course know that we must only use the structures introduced there. Therefore, since we want to implement a register with asynchronous reset, we create a synchronous process sensitive to the clock and reset signal.

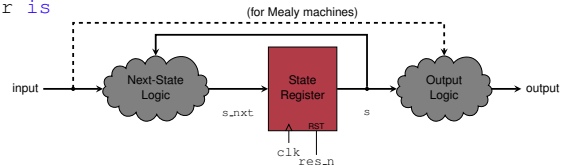
Example: State Register

4 Implement the state register in a sync. process

```

18 architecture arch of advanced_timer is
19   -- enum and record declarations
20   signal s, s_nxt : state_reg_t;
21 begin
22   state_reg : process(clk, res_n)
23   begin
24     if res_n = '0' then
25       s <= (state => IDLE, others => (others => '0'));
26     elsif rising_edge(clk) then
27       s <= s_nxt;
28     end if;
29   end process;
30 end arch;

```



Finite-State Machine Implementation (in VHDL)

Example

Example: State Register

4 Implement the state register in a sync. process

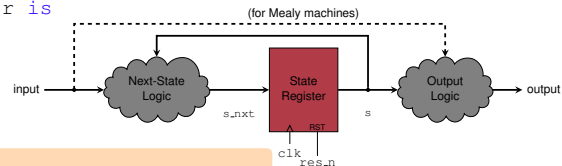


Next, we implement the asynchronous reset functionality. From the model we know that the FSM should start in the IDLE state. Remember that it is good practice to always provide reset values for all flip-flops in a design. Hence, we set the two counters to zero. The actual value for these counters doesn't really matter, as they are initialized to zero in the IDLE state anyway. Please note that this coding style for FSMs enforces concrete reset values for all elements of the state register at compile time. If you forget to provide a value of a particular element in the aggregate expression representing the reset value, the code compilation fails.

Example: State Register

4 Implement the state register in a sync. process

```
18 architecture arch of advanced_timer is
19   -- enum and record declarations
20   signal s, s_nxt : state_reg_t;
21 begin
22   state_reg : process(clk, res_n)
23   begin
24     if res_n = '0' then
25       s <= (state => IDLE, others => (others => '0'));
26     elsif rising_edge(clk) then
27       s <= s_nxt;
28     end if;
29   end process;
30 end architecture;
```



HWMMod
WS25

FSM Implementation
Implementation
Steps
Example
FSM Types
State Register
Comb. Logic

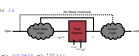
Finite-State Machine Implementation (in VHDL)

Example

Example: State Register

4 Implement the state register in a sync. process

```
18 architecture arch of advanced_timer is
19 -- enum and record declarations
20 signal s, s_nxt : state_reg_t;
21 begin
22 state_reg : process(clk, res_n)
23 begin
24 if res_n = '0' then
25 s <= (state => IDLE, others => (others => '0'));
26 elsif rising_edge(clk) then
27 s <= s_nxt;
28 end if;
29 end process;
30 end arch;
```

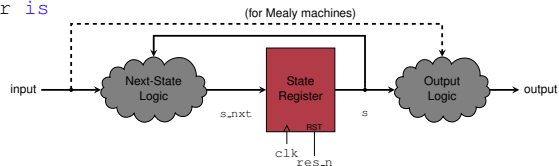


At each active clock edge, we simply assign the next-state signal to the current state signal. Since we are using a record type for the state register, this can be done in a single line of code. With that we are done implementing the state register.

Example: State Register

4 Implement the state register in a sync. process

```
18 architecture arch of advanced_timer is
19 -- enum and record declarations
20 signal s, s_nxt : state_reg_t;
21 begin
22 state_reg : process(clk, res_n)
23 begin
24 if res_n = '0' then
25 s <= (state => IDLE, others => (others => '0'));
26 elsif rising_edge(clk) then
27 s <= s_nxt;
28 end if;
29 end process;
30 end arch;
```



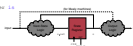
Finite-State Machine Implementation (in VHDL)

Example

Example: State Register

4 Implement the state register in a sync. process

```
18 architecture arch of advanced_timer is
19 -- enum and record declarations
20 signal s, s_nxt : state_reg_t;
21 begin
22 state_reg : process(clk, res_n)
23 begin
24 if res_n = '0' then
25 s <= (state => IDLE, others => (others => '0'));
26 elsif rising_edge(clk) then
27 s <= s_nxt;
28 end if;
29 end process;
30
```

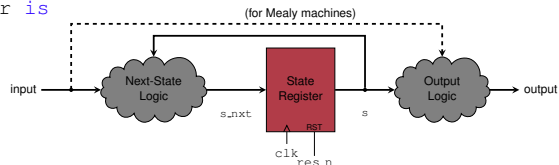


Note how the code for our state register is, except for the reset, completely independent of the particular FSM we implement because we use the dedicated record type. At this point we want to point out that this is **always** the only code required to implement the state register of an FSM and that you can use this general structure for all FSMs in our courses. We will now look at the implementation of the next-state and output functions.

Example: State Register

4 Implement the state register in a sync. process

```
18 architecture arch of advanced_timer is
19 -- enum and record declarations
20 signal s, s_nxt : state_reg_t;
21 begin
22 state_reg : process(clk, res_n)
23 begin
24 if res_n = '0' then
25 s <= (state => IDLE, others => (others => '0'));
26 elsif rising_edge(clk) then
27 s <= s_nxt;
28 end if;
29 end process;
30
```



The final step in the FSM implementation flow is about the next-state and output logic. Since these two combinational logic blocks essentially define the behavior of the FSM, this is also the most elaborate step. However, by basing the code on the FSM model again, this step is nevertheless usually straight forward. Since we are using the 2-process approach, we will implement the two functions in a single combinational process. The slide already shows its first few lines. However, before we continue let us address one of the most common mistakes we observed beginners make when implementing FSMs.

Example: Next-State and Output Logic

5 Implement the next-state and output logic in comb. process

```
36 comb : process(all)
37 begin
38     s_next <= s;
39     hex <= to_segs(std_ulogic_vector(s.tick_cnt));
```

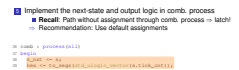
Recall what we heard about inferred latches in a previous lecture: A latch is inferred in a combinational process if there exists both, a path which writes to some signal, and one in which it does not write to the same signal. When implementing FSMs this often accidentally happens when either an output or an element of the next state is not assigned a value within the respective process. However, by adhering to the flow shown in this lecture it is easy to mitigate such mistakes in the first place. You know that, by design, only the `s_next` signal or an FSM output can result in a latch. Since these signals are only driven by the FSMs combinational process, you can easily ensure that they are always assigned a value.

Example: Next-State and Output Logic

- 5 Implement the next-state and output logic in comb. process
 - **Recall:** Path without assignment through comb. process ⇒ latch!

```

36 comb : process(all)
37 begin
38     s_next <= s;
39     hex <= to_segs(std_ulogic_vector(s.tick_cnt));
    
```



To achieve this we recommend you to do as shown on the slide. The first thing you should do in your combinational process is to assign `s` to the `s_next` signal. Recall that, in the modeling lecture, we defined that the state register simply keeps its value if there are no state updates performed in a state. This is exactly, what is expressed using this default assignment. Then you should provide a default value for each of your outputs. This default value can either depend on the current value of the state register, an input or a constant. In our example we use the current value of `tick_cnt` converted to a sensible seven segment display value. Since two of the three states use this assignment anyway, this is a reasonable choice that reduces the amount of code we need to write for the individual states.

Example: Next-State and Output Logic

- 5 Implement the next-state and output logic in comb. process
 - **Recall:** Path without assignment through comb. process ⇒ latch!
⇒ Recommendation: Use default assignments

```
36 comb : process(all)
37 begin
38   s_next <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
```

After providing default assignments we can implement the actual next-state and output logic. For that we can simply use a `case` statement and define the two logic functions per state. This way the code does not become too complex, is easy to read, modify and maintain. Furthermore, as before, this general code structure is the same for all FSMs. Let us now look at the two logic functions for the IDLE state.

Example: Next-State and Output Logic

- 5 Implement the next-state and output logic in comb. process
 - **Recall:** Path without assignment through comb. process => latch!
 - ⇒ Recommendation: Use default assignments
 - Define next-state and output logic per state

```

36 comb : process(all)
37 begin
38   s_next <= s;
39   hex <= to_seg8(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is

```

Finite-State Machine Implementation (in VHDL)

Example

Example: Next-State and Output Logic

- Implement the next-state and output logic in comb. process
 - Recall: Path without assignment through comb. process ⇒ latch!
 - ⇒ Recommendation: Use default assignments
 - Define next-state and output logic per state

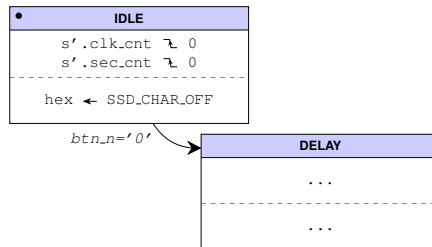


To implement the combinational logic required for a state, we need the respective state node of our model, as well as all out-going edges. For the IDLE state this is shown on the slide.

Example: Next-State and Output Logic

- 5 Implement the next-state and output logic in comb. process
 - **Recall:** Path without assignment through comb. process ⇒ latch!
 - ⇒ Recommendation: Use default assignments
 - Define next-state and output logic per state

```
36 comb : process(all)
37 begin
38   s_next <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
```



HWMMod
WS25

FSM Implementation
Implementation Steps
Example
FSM Types
State Register
Comb. Logic

Finite-State Machine Implementation (in VHDL)

Example

Example: Next-State and Output Logic

- Implement the next-state and output logic in comb. process
- Recall: Path without assignment through comb. process ⇒ latch!
- Recommendation: Use default assignments
- Define next-state and output logic per state

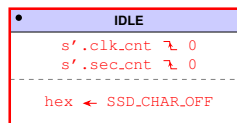


Inside the **when** clause we write the state updates and output actions of the state node if they are not covered by the default assignments. In our case all actions inside the IDLE state node differ from the default assignments.

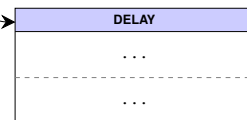
Example: Next-State and Output Logic

- 5 Implement the next-state and output logic in comb. process
 - **Recall:** Path without assignment through comb. process ⇒ latch!
 - ⇒ Recommendation: Use default assignments
 - Define next-state and output logic per state

```
36 comb : process(all)
37 begin
38   s_nxt <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
43       s_nxt.clk_cnt <= (others => '0');
44       s_nxt.tick_cnt <= (others => '0');
45       hex <= SSD_CHAR_OFF;
```



btn_n='0'



HWMMod
WS25

FSM Implementation

Implementation Steps

Example

FSM Types

State Register

Comb. Logic

Finite-State Machine Implementation (in VHDL)

Example

Example: Next-State and Output Logic

- Implement the next-state and output logic in comb. process
- Recall: Path without assignment through comb. process ⇒ latch!
- Recommendation: Use default assignments
- Define next-state and output logic per state

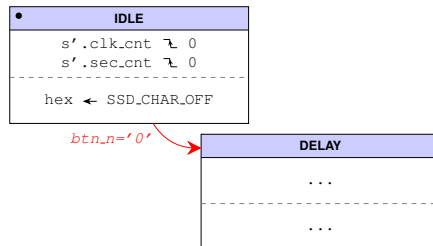


Next, we add an `if` statement for all outgoing edges. The condition of each `if` statement is the condition of the respective edge, whereas inside the body the destination state of the transition edge is set to be the next FSM state. Furthermore, all actions of the respective edge are also added to this body. In our case there are no transition edges.

Example: Next-State and Output Logic

- Implement the next-state and output logic in comb. process
 - Recall:** Path without assignment through comb. process ⇒ latch!
 - ⇒ Recommendation: Use default assignments
 - Define next-state and output logic per state

```
36 comb : process(all)
37 begin
38   s_nxt <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
43       s_nxt.clk_cnt <= (others => '0');
44       s_nxt.tick_cnt <= (others => '0');
45       hex <= SSD_CHAR_OFF;
46       if btn_n = '0' then
47         s_nxt.state <= DELAY;
48       end if;
49   end case;
```



HWMMod
WS25

FSM Implementation

Implementation Steps

Example

FSM Types

State Register

Comb. Logic

Finite-State Machine Implementation (in VHDL)

Example

Example: Next-State and Output Logic

- Implement the next-state and output logic in comb. process
- Recall: Path without assignment through comb. process ⇒ latch!
- Recommendation: Use default assignments
- Define next-state and output logic per state

```
36 comb : process(all)
37 begin
38   s_nxt <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
43       s_nxt.clk_cnt <= (others => '0');
44       s_nxt.tick_cnt <= (others => '0');
45       hex <= SSD_CHAR_OFF;
46       if btn_n = '0' then
47         s_nxt.state <= DELAY;
48       end if;
49   end case;
```

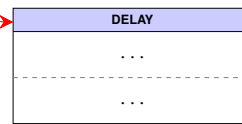
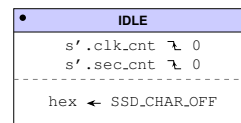


We continue in this manner, defining the next-state and output logic functions for all states of our FSM.

Example: Next-State and Output Logic

- Implement the next-state and output logic in comb. process
 - Recall: Path without assignment through comb. process ⇒ latch!
 - ⇒ Recommendation: Use default assignments
 - Define next-state and output logic per state

```
36 comb : process(all)
37 begin
38   s_nxt <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
43       s_nxt.clk_cnt <= (others => '0');
44       s_nxt.tick_cnt <= (others => '0');
45       hex <= SSD_CHAR_OFF;
46       if btn_n = '0' then
47         s_nxt.state <= DELAY;
48       end if;
49   end case;
```



Finite-State Machine Implementation (in VHDL)

Example

Example: Next-State and Output Logic (cont'd)

```
36 comb : process(all)
37 begin
38   s_nxt <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
43       s_nxt.clk_cnt <= (others => '0');
44       s_nxt.tick_cnt <= (others => '0');
45       hex <= SSD_CHAR_OFF;
46       if btn_n = '0' then
47         s_nxt.state <= DELAY;
48       end if;
49
50     when DELAY =>
51       if s.clk_cnt < CLK_FREQ-1 then
52         s_nxt.clk_cnt <= s.clk_cnt + 1;
53       else
54         s_nxt.state <= TICK;
55       end if;
56
57     when TICK =>
58       s_nxt.clk_cnt <= (others => '0');
59       s_nxt.state <= DELAY;
60       if s.tick_cnt < 15 then
61         s_nxt.tick_cnt <= s.tick_cnt + 1;
62       else
63         s_nxt.state <= IDLE;
64       end if;
65   end case;
66 end process;
```

The result of this is shown on this slide. Note how the code features three `when` statements - one for each abstract FSM state. Furthermore, observe how all states have the same structure and how the `otherwise` edges of the `DELAY` and `TICK` states simply correspond to `else` statements.

Example: Next-State and Output Logic (cont'd)

```
36 comb : process(all)
37 begin
38   s_nxt <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
43       s_nxt.clk_cnt <= (others => '0');
44       s_nxt.tick_cnt <= (others => '0');
45       hex <= SSD_CHAR_OFF;
46       if btn_n = '0' then
47         s_nxt.state <= DELAY;
48       end if;
49
50     when DELAY =>
51       if s.clk_cnt < CLK_FREQ-1 then
52         s_nxt.clk_cnt <= s.clk_cnt + 1;
53       else
54         s_nxt.state <= TICK;
55       end if;
56
57     when TICK =>
58       s_nxt.clk_cnt <= (others => '0');
59       s_nxt.state <= DELAY;
60       if s.tick_cnt < 15 then
61         s_nxt.tick_cnt <= s.tick_cnt + 1;
62       else
63         s_nxt.state <= IDLE;
64       end if;
65   end case;
66 end process;
```

HWMoD
WS25

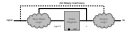
FSM Imple-
mentation
Implementation
Steps
Example
FSM Types
State Register
Comb. Logic

Finite-State Machine Implementation (in VHDL)

Example

Example: Next-State and Output Logic (cont'd)

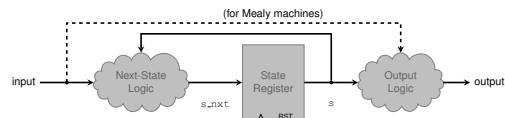
```
36 comb : process(all)
37 begin
38   s_nxt <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
43       s_nxt.clk_cnt <= (others => '0');
44       s_nxt.tick_cnt <= (others => '0');
45       hex <= SSD_CHAR_OFF;
46       if btn_n = '0' then
47         s_nxt.state <= DELAY;
48       end if;
49
50     when DELAY =>
51       if s.clk_cnt < CLK_FREQ-1 then
52         s_nxt.clk_cnt <= s.clk_cnt + 1;
53       else
54         s_nxt.state <= TICK;
55       end if;
56
57     when TICK =>
58       s_nxt.clk_cnt <= (others => '0');
59       s_nxt.state <= DELAY;
60       if s.tick_cnt < 15 then
61         s_nxt.tick_cnt <= s.tick_cnt + 1;
62       else
63         s_nxt.state <= IDLE;
64       end if;
65   end case;
66 end process;
```



Finally, let us look at where we can find the next-state and output logic inside this single combinational process.

Example: Next-State and Output Logic (cont'd)

```
36 comb : process(all)
37 begin
38   s_nxt <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
43       s_nxt.clk_cnt <= (others => '0');
44       s_nxt.tick_cnt <= (others => '0');
45       hex <= SSD_CHAR_OFF;
46       if btn_n = '0' then
47         s_nxt.state <= DELAY;
48       end if;
49
50     when DELAY =>
51       if s.clk_cnt < CLK_FREQ-1 then
52         s_nxt.clk_cnt <= s.clk_cnt + 1;
53       else
54         s_nxt.state <= TICK;
55       end if;
56
57     when TICK =>
58       s_nxt.clk_cnt <= (others => '0');
59       s_nxt.state <= DELAY;
60       if s.tick_cnt < 15 then
61         s_nxt.tick_cnt <= s.tick_cnt + 1;
62       else
63         s_nxt.state <= IDLE;
64       end if;
65   end case;
66 end process;
```



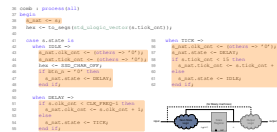
HWMMod
WS25

FSM Implementation
Implementation Steps
Example
FSM Types
State Register
Comb. Logic

Finite-State Machine Implementation (in VHDL)

Example

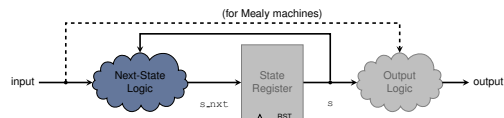
Example: Next-State and Output Logic (cont'd)



The next-state logic comprises all assignments to the `s_nxt` signal, including respective conditional statements and the default assignments.

Example: Next-State and Output Logic (cont'd)

```
36 comb : process(all)
37 begin
38   s_nxt <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
43       s_nxt.clk_cnt <= (others => '0');
44       s_nxt.tick_cnt <= (others => '0');
45       hex <= SSD_CHAR_OFF;
46       if btn_n = '0' then
47         s_nxt.state <= DELAY;
48       end if;
49
50     when DELAY =>
51       if s.clk_cnt < CLK_FREQ-1 then
52         s_nxt.clk_cnt <= s.clk_cnt + 1;
53       else
54         s_nxt.state <= TICK;
55       end if;
56
57     when TICK =>
58       s_nxt.clk_cnt <= (others => '0');
59       s_nxt.state <= DELAY;
60       if s.tick_cnt < 15 then
61         s_nxt.tick_cnt <= s.tick_cnt + 1;
62       else
63         s_nxt.state <= IDLE;
64       end if;
65   end case;
66 end process;
```



HWMMod
WS25

FSM Implementation
Implementation Steps
Example
FSM Types
State Register
Comb. Logic

Finite-State Machine Implementation (in VHDL)

Example

Example: Next-State and Output Logic (cont'd)

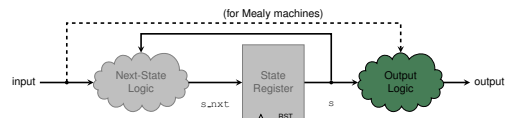
```
36 comb : process(all)
37 begin
38   s_nxt <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
43       s_nxt.clk_cnt <= (others => '0');
44       s_nxt.tick_cnt <= (others => '0');
45       hex <= SSD_CHAR_OFF;
46       if btn_n = '0' then
47         s_nxt.state <= DELAY;
48       end if;
49
50     when DELAY =>
51       if s.clk_cnt < CLK_FREQ-1 then
52         s_nxt.clk_cnt <= s.clk_cnt + 1;
53       else
54         s_nxt.state <= TICK;
55       end if;
56
57     when TICK =>
58       s_nxt.clk_cnt <= (others => '0');
59       s_nxt.state <= DELAY;
60       if s.tick_cnt < 15 then
61         s_nxt.tick_cnt <= s.tick_cnt + 1;
62       else
63         s_nxt.state <= IDLE;
64       end if;
65   end case;
66 end process;
```



Similarly, the output logic comprises the assignments to the FSM outputs, as well as respective conditional statements and default assignments. In this example, due to the sensible choice of the default value for `hex` only very little code is required for the output logic. Finally, please note that the left-hand side of all assignments in the process only use the next state signal or an output. The right-hand sides, as well as expressions used in if-conditions, only use the state signals or inputs. This property should hold up in most FSMs you design and implement! This concludes this lecture about the implementation of FSMs in VHDL. The key take-aways of this video are that coding FSMs is a simple task if you use our FSM model and adhere to the steps introduced in this lecture.

Example: Next-State and Output Logic (cont'd)

```
36 comb : process(all)
37 begin
38   s_nxt <= s;
39   hex <= to_segs(std_ulogic_vector(s.tick_cnt));
40
41   case s.state is
42     when IDLE =>
43       s_nxt.clk_cnt <= (others => '0');
44       s_nxt.tick_cnt <= (others => '0');
45       hex <= SSD_CHAR_OFF;
46       if btn_n = '0' then
47         s_nxt.state <= DELAY;
48       end if;
49
50     when DELAY =>
51       if s.clk_cnt < CLK_FREQ-1 then
52         s_nxt.clk_cnt <= s.clk_cnt + 1;
53       else
54         s_nxt.state <= TICK;
55       end if;
56
57     when TICK =>
58       s_nxt.clk_cnt <= (others => '0');
59       s_nxt.state <= DELAY;
60       if s.tick_cnt < 15 then
61         s_nxt.tick_cnt <= s.tick_cnt + 1;
62       else
63         s_nxt.state <= IDLE;
64       end if;
65   end case;
66 end process;
```



Thank you for listening! We recommend you to immediately take the self-check test in TUWEL, to see if you understood the material presented in this lecture.

HWMod
WS25

FSM Imple-
mentation

Implementation
Steps

Example

FSM Types

State Register

Comb. Logic

Lecture Complete!